Attorney's Docket No.: 6570P012

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Application for:

Frank Killian

Serial No. 10/749,543

Filed: 12-30-2003

For: SYSTEM AND METHOD FOR MONITORING AND CONTROLLING SERVER NODES CONTAINED WITHIN A CLUSTERED ENVIRONMENT

Examiner: PANTOLIANO JR., RICHARD

Art Unit: 2194

Mail Stop Appeal Brief – Patents
Commissioner For Patents
P.O. Box 1450
Alexandria, V.A. 22313-1450

APPEAL BRIEF

Dear Sir:

The Applicant ("Appellant") submits the following Appeal Brief pursuant to 37 C.F.R. §41.37(c) for consideration by the Board of Patent Appeals and Interferences. A payment in the amount of $510.00 was submitted with the Notice of Appeal filed on February 27, 2008, (received by the Patent Office on March 3, 2008) as required by 37 C.F.R. §41.20(b)(1). A payment in the amount of $510.00 is submitted herewith as required by 37 C.F.R. §41.20(b)(2).

TABLE OF CONTENTS

I. REAL PARTY IN INTEREST

Frank Killian is named as the inventor on the application. Frank Killian transferred his rights in the subject application through an assignment executed on December 30, 2003, to SAP Aktiengesellschaft ("SAP AG"), a Corporation of Germany, having a principal place of business at Walldorf, Germany. The assignment is recorded at reel/frame number 014866/0388. Accordingly, SAP AG is the real party in interest.

II. RELATED APPEALS AND INTERFERENCES

There are no other appeals or interferences that will directly affect, be directly affected by or have a bearing on the Board's decision in this Appeal.

III. STATUS OF CLAIMS

Claims 1-39 are pending in the application. The Examiner has rejected claims 1-39. Appellant respectfully appeals the rejection of claims 1-39.

IV. STATUS OF AMENDMENTS

No amendments were submitted after the Final Office Action dated November 27, 2007.

V. SUMMARY OF CLAIMED SUBJECT MATTER

Embodiments of the invention relate to a system and method for monitoring and controlling server nodes contained within a clustered environment. See Specification, page 1, paragraph [0001], lines 6-8.

In regard to independent claim 1, the system includes a cluster having a first instance and a second instance, each of the first and second instances including a plurality of server nodes. See Specification, page 3, paragraph [00011], lines 5-9 and 19-23; Fig. 1, elements 100, 112-1 to 112-N; Fig. 2, elements 202, and 210. A control logic starts each instance by initiating a launch logic for each of the server nodes, and the launch logic, when initiated, executes Java processes in each respective server node. See Specification, page 4, paragraph [00015], lines 11 and 12 and 19-22; Fig. 3, elements 305,

335, 357. A communication interface is coupled between the launch logic and the control logic to enable the launch logic to obtain status of each of the Java processes and enable the control logic to access the status in a shared memory via the communication interface. See Specification, page 4, paragraph [00017], lines 30-32; page 5, paragraph [00018], lines 6-9; Fig 3, elements 305, 325, and 335-1 to 335-N. Finally, the launch logic stores and maintains the status in the shared memory via the communication interface. See Specification, page 4, paragraph [00017], line 34 to page 5, line 3; Fig. 3, elements 325, 330-1 to 330-N, 355-1 to 355-N, 335, and 305.

Claim 2 depends on claim 1 and includes the further limitations of the launch logic loading a virtual machine and executing a Java process in the virtual machine. See Specification, page 4, paragraph [00016], lines 19-22; Fig. 3, elements 335-1, 357 and 355-1.

Claim 3 depends on claim 2 and includes the further limitations of the communication interface as the shared memory to store the status of the Java processes. See Specification, page 4, paragraph [00017], line 32 to page 5, line 5; Fig. 3, elements 325, 355-1 to 355-N.

Claim 4 depends on claim 3 and includes the further limitations of the launch logic as a Java native interface to obtain the status of each of the Java processes and to update the shared memory with the obtained status. See Specification, page 5, paragraph [00018], lines 9-17; Fig. 3, elements 350, 335, 355, 325.

Claim 5 depends on claim 4 and recites the further limitations of the control logic accesses the shared memory to monitor the status of each of the Java processes. See Specification, page 5, paragraph [00019], lines 19-23; Fig. 3, elements 305, 325, and 355.

Claim 6 depends on claim 1 and recites the further limitations of the control logic detects a failure of a Java process and to automatically restart the failed Java process. See Specification, page 7, paragraph [00026], lines 19-26; Fig. 3, elements 305 and 355.

Claim 7 depends on the system of claim 1 and recites the additional limitations of the control logic generates an instruction to start, terminate or restart a particular process executed server nodes based on a command received from a remote device. See

Specification, page 6, paragraph [00021], lines 2-12; Fig. 3, elements 305, 355, 250, and 310.

Claim 8 depends on claim 1 and recites the further limitations of the communication interface as a named pipe to send and receive commands between the control logic and the launch logic. See Specification, page 6, paragraph [00023], lines 23-25; Fig. 3, elements 345, 305, and 335.

Claim 9 depends on claim 1 and includes the further limitations of the control logic as a signal handler to receive and interpret signals from a management console. See Specification, page 6, paragraph [00021], lines 3 and 4; Fig. 3, elements 310, 305, and 250.

Claim 10 depends on claim 1 and recites the further limitations of the control logic as a server connector to enable connection with an external server. See Specification, page 6, paragraph [00022], lines 13-15; Fig. 3, elements 305, 220, and 320.

Claim 11 depends on claim 1 and recites the further limitations of the control logic as Java native processes. See Specification, page 5, paragraph [00017], lines 1-3; paragraph [00018], lines 9-14; paragraph [00019], lines 21-23; Fig. 3, elements 305 and 355.

Claim 12 depends on claim 1 and recites the additional limitations of the launch logic as a container combining Java native processes with a Java virtual machine. See Specification, page 4, paragraph [00016], lines 26-29; Fig. 3, elements 335, 355, and 357.

Independent claim 13 recites a method with the limitations of executing Java processes for a plurality of server nodes in an instance. See Specification, page 4, paragraph [00015], lines 8-11; Fig. 1; Fig. 2, elements 112 and 202; Fig. 3, elements 300, 305, 335, and 355. The status is obtained regarding the Java processes executed by the server nodes in the instance. See Specification, page 4, paragraph [00017], lines 30-32; page 5, paragraph [00018], lines 6-9; Fig. 1, elements 112 and 202; Fig 3, elements 305, 325, and 335-1 to 335-N. The status is stored regarding the Java processes in a communication interface, the communication interface updating and maintaining the status in a shared memory. See Specification, page 4, paragraph [00017], line 34 to page

5, line 3; Fig. 3, elements 325, 330-1 to 330-N, 355-1 to 355-N, 335, and 305. Finally, the status is accessed in the shared memory via the communication interface. See Specification, page 5, paragraph [00018], lines 6-9; Fig. 3, elements 355, 335, 357, 305, and 325.

Claim 14 depends on claim 13 and recites the further limitations of enabling control of the Java processes based on an instruction received from a remote device. See Specification, page 6, paragraph [00021], lines 7-12; Fig. 3, elements 250, 355, and 310.

Claim 15 depends on claim 13 and recites the further limitations of using a Java native interface to obtain the status regarding the Java processes. See Specification, page 5, paragraph [00018], lines 9-17; Fig. 3, elements 350, 335, 355, and 325.

Claim 16 depends on claim 13 and recites the additional limitations of detecting a failure of a process within the cluster by accessing the status in the communication interface and restarting the failed process. See Specification, page 7, paragraph [00026], lines 19-26; Fig. 3, elements 305, 325, and 355.

Independent claim 17 recites a machine-readable medium that provides instructions comprising executing Java processes for a plurality of server nodes in an instance. See Specification, page 4, paragraph [00015], lines 11 and 12 and 19-22; Fig. 3, elements 305, 335, 357. Status is obtained regarding each of the Java processes executed by the server nodes in the instance. See Specification, page 4, paragraph [00017], lines 30-32; page 5, paragraph [00018], lines 6-9; Fig. 1, elements 112 and 202; Fig 3, elements 305, 325, and 335-1 to 335-N. The status is stored regarding the Java processes into a shared memory. See Specification, page 4, paragraph [00017], line 34 to page 5, line 3; Fig. 3, elements 325, 330-1 to 330-N, 355-1 to 355-N, 335, and 305.

Claim 18 depends on claim 17 and recites the further limitations of invoking a Java native interface to obtain the status regarding the Java processes. See Specification, page 5, paragraph [00018], lines 9-17; Fig. 3, elements 350, 335, 355, and 325.

Claim 19 depends on claim 17 and recites the further limitations of receiving instructions via a communication interface. See Specification, page 6, paragraph [00021], lines 3 and 4; Fig. 3, elements 310, 305, and 250. Claim 19 recites the additional

limitations of starting, terminating or restarting a process based on the instructions received via the communication interface. See Specification, page 6, paragraph [00021], lines 2-12; Fig. 3, elements 305, 355, 250, and 310.

Claim 20 depends on claim 17 and recites the further limitations of detecting a failure of a process within the cluster by accessing the status in the shared memory and automatically restarting the failed process. See Specification, page 7, paragraph [00026], lines 19-26; Fig. 3, elements 305 and 355.

Independent claim 21 recites an apparatus comprising a cluster having a first instance and a second instance, each of the first and second instances including a plurality of server nodes. See Specification, page 3, paragraph [00011], lines 5-9 and 19-23; Fig. 1, elements 100, 112-1 to 112-N; Fig. 2, elements 202, and 210. A control logic starts each respective instance by initiating a launch logic for each respective server node in the first and second instances. See Specification, page 4, paragraph [00015], lines 11 and 12 and 19-22; Fig. 3, elements 305, 335, 357. The launch logic, for each respective server node in the first and second instances, to further launch Java processes, and obtain a status of the Java processes to store and maintain in a shared memory. See Specification, page 4, paragraph [00017], line 34 to page 5, line 3; Fig. 3, elements 325, 330-1 to 330-N, 355-1 to 355-N, 335, and 305. Finally, the control logic accesses the status obtained by the launch logic via the shared memory. See Specification, page 4, paragraph [00017], lines 30-32; page 5, paragraph [00018], lines 6-9; Fig 3, elements 305, 325, and 335-1 to 335-N.

Claim 22 depends on claim 21 and recites the additional limitations of the shared memory to enable exchange of information between the Java processes and the control logic. See Specification, page 4, paragraph [00017], line 34 to page 5, line 3; Fig 3, elements 325, 355, 305.

Claim 23 depends on claim 21 and recites the additional limitations of the launch logic loading a virtual machine and executing Java processes. See Specification, page 4, paragraph [00016], lines 19-22; Fig. 3, elements 335-1, 357 and 355-1.

Claim 24 depends on claim 23 and recites the additional limitations of the launch logic using a Java native interface to obtain a status of each of the Java processes and updates information contained in the shared memory based on the status obtained by the Java native interface. See Specification, page 5, paragraph [00018], lines 9-17; Fig. 3, elements 350, 335, 355, 325.

Claim 25 depends on claim 21 and recites the additional limitations of the control logic detects a failure of a process within the cluster and automatically restarts operations of the failed process. See Specification, page 7, paragraph [00026], lines 19-26; Fig. 3, elements 305 and 355.

Claim 26 depends on claim 21 and recites the limitations of a signal handler to receive a command from a remote device and controlling one of the Java processes based on the command received from the remote device. See Specification, page 6, paragraph [00021], lines 3 and 4; Fig. 3, elements 310, 305, and 250.

Claim 27 depends on claim 21 and recites the further limitations of a named pipe to send and receive commands between the control logic and the launch logic. See Specification, page 6, paragraph [00023], lines 23-25; Fig. 3, elements 345, 305, and 335.

Independent claim 28 recites a system comprising a cluster having a first instance and a second instance, each of the first and second instances including a plurality of server nodes. See Specification, page 3, paragraph [00011], lines 5-9 and 19-23; Fig. 1, elements 100, 112-1 to 112-N; Fig. 2, elements 202, and 210. Claim 28 further recites a means for starting each instance by executing Java processes in each respective server node. See Specification, page 4, paragraph [00015], lines 11 and 12 and 19-22; Fig. 3, elements 305, 335, 357. Claim 28 recites a means for enabling exchange of information that is stored and maintained in a shared memory between the Java processes and the means for starting each instance. See Specification, page 4, paragraph [00017], line 34 to page 5, line 3; Fig 3, elements 325, 355, 305.

Claim 29 depends on claim 28 and recites the further limitations of a means for loading a virtual machine and execute a Java process in the virtual machine. See

Specification, page 4, paragraph [00016], lines 19-22; Fig. 3, elements 335-1, 357 and 355-1.

Claim 30 depends on claim 28 and recites the additional limitations of the means for enabling exchange of information comprises the shared memory having a plurality of entries. See Specification, page 4, paragraph [00017], line 32 to page 5, line 3; Fig. 3, elements 305, 330, 325, 335, and 355.

Claim 31 depends on claim 30 and recites the further limitations of a means for obtaining status for each of the Java processes and a means for updating the shared memory with the obtained status. See Specification, page 5, paragraph [00018], lines 9-17; Fig. 3, elements 350, 335, 355, 325.

Claim 32 depends on claim 31 and recites the additional limitations of a means for accessing the shared memory to monitor the status of each of the Java processes. See Specification, page 5, paragraph [00019], lines 19-23; Fig. 3, elements 305, 325, and 355. Claim 32 recites the further limitations of a means for sending an instruction to the launch means to start, terminate or restart a particular process executed in the cluster. See Specification, page 6, paragraph [00021], lines 2-12; Fig. 3, elements 305, 355, 250, and 310.

Claim 33 depends on claim 28 and recites the further limitations of a means for enabling a user to monitor and control the Java processes running in the cluster from a management console coupled to the means for controlling. See Specification, page 6, paragraph [00021], lines 1-7; Fig. 3, elements 305, 250, and 310. Claim 33 recites the further limitations of a means for enabling a connection with an external server. See Specification, page 6, paragraph [00022], lines 13-15; Fig. 3, elements 305, 320, and 220.

Claim 34 depends on claim 1 and recites the additional limitations of the launch logic stores the status independent of the control logic accessing the status. See Specification, page 4, paragraph [00017], line 30 to page 5, paragraph [00019], line 25; Fig. 3, elements 335, 305, 325, 350, and 315.

Claim 35 depends on claim 1 and recites the further limitations of a persistent data structure is stored in the shared memory to enable an independent exchange of information between the Java processes. See Specification, page 4, paragraph [00017], line 30 to page 5, paragraph [00019], line 25; Fig. 3, elements 335, 305, 325, 350, and 315.

Claim 36 depends on claim 13 and recites the limitations of accessing the status occurs independent of storing the status. See Specification, page 4, paragraph [00017], line 30 to page 5, paragraph [00019], line 25; Fig. 3, elements 335, 305, 325, 350, and 315.

Claim 37 depends on claim 17 and recites the further limitations of storing the status occurs independent of accessing the status. See Specification, page 4, paragraph [00017], line 30 to page 5, paragraph [00019], line 25; Fig. 3, elements 335, 305, 325, 350, and 315.

Claim 38 depends on claim 21 and recites the additional limitations of the control logic accesses the status independent of the launch logic storing the status. See Specification, page 4, paragraph [00017], line 30 to page 5, paragraph [00019], line 25; Fig. 3, elements 335, 305, 325, 350, and 315.

Claim 39 depends on claim 28 and recites the further limitations of the exchange of information is achieved by an independent storing and an independent accessing of the information. See Specification, page 4, paragraph [00017], line 30 to page 5, paragraph [00019], line 25; Fig. 3, elements 335, 305, 325, 350, and 315.

VI. GROUNDS OF REJECTION TO BE REVIEWED ON APPEAL

In the Final Office Action dated November 27, 2007, the Examiner has rejected claims 1-39.

Claims 1-3, 6, 7, 9, 10, 13, 14, 16, 17, 19-23, 25, 26, and 28-39 stand rejected under 35 U.S.C. § 103(a) as being unpatentable over U.S. Patent Publication No. 2005/0005200 filed by Matena et al. (hereinafter "Matena") in view of U.S. Patent No. 5,991,893 issued to Snider (hereinafter "Snider").

Claims 4, 5, 8, 11, 12, 15, 18, 24, and 27 stand rejected under 35 U.S.C. § 103(a) as being obvious over Matena in view Snider in further view of U.S. Patent No. 6,823,358 issued to Spender (hereinafter "Spender").

All of the claims do not stand or fall together. The basis for the separate patentability of the claims is set forth below.

VII. ARGUMENT

A. Overview of the Prior Art -- Matena

Matena discloses a system for controlling the execution of applications in a distributed computing system. See Matena, paragraph [0078]. The system includes a node controller, execution controller, service application controller, Java application controller, and a customer application controller. See Matena, paragraph [0083]. The node controller starts and stops processes on the node. See Matena, paragraph [0078]. An execution controller maintains information about the distributed computing system related to the nodes of different computer architectures and operating systems, along with the processes that are running on the nodes. Id. An application controller (e.g., service or Java type) starts applications, stops applications, load-balances applications over the nodes, recovers applications from failures, and upgrades applications. Id.

Further, Matena discloses that an application controller supports various operations in an API so that external applications may invoke the operations. See Matena, paragraphs [0214] and [0221]. In addition, nodes may be natively implemented (e.g., using C or C++) and nodes communicate with each other via shared memory. See Matena, paragraph [0413].

However, Matena is silent on the content and type of communication that takes place between the nodes. Id. Moreover, Matena fails to at least disclose the location of an interface for communication between a control logic and a launch logic. Matena also fails to at least disclose the elements of the control logic to access the status in a shared memory via a communication interface. Matena further fails to disclose the elements of the launch logic to store and maintain the status in the shared memory via the communication interface.

B. Overview of the Prior Art -- Snider

Snider discloses a system that provides a software layer between a single operating system and underlying nodes to handle memory sharing and implement fail-safe protocols for the underlying nodes. See Snider, column 3, line 67 to column 4, line 9. The software layer may include data structures that are used by the operating system and underlying nodes. See Snider, column 5, lines 2-7. To allow the nodes to access the data structure in the software layer, Snider provides a locking mechanism to requires that the nodes access the data structure in a serial manner. See Snider, column 6, line 64 to column 7, line 9. In this manner, Snider discloses a dependent scheme for accessing and exchanging information in the data structure that ensures the integrity of the data in the data structure. See Snider, column 7, lines 1-7.

However, Snider at least fails to disclose the control logic to access the status in a shared memory via the communication interface, the status being of Java processes running on the server nodes in a cluster. In addition, Snider fails to disclose that storing the status is independent of accessing the status.

C. Overview of the Prior Art -- Spender

Spender discloses a system for distinguishing between requests sent from Java client applications that are identified using a shared process ID. See Spender, column 3, line 55 to column 4, line 1. To accomplish this goal, each application is assigned to a respective dispatcher application having a unique process ID and then routing all requests through that uniquely identified dispatcher application. See Spender, column 4, lines 1-5. A manager application running on the same machine as the Java client applications is responsible for launching and allocating the dispatcher applications to each Java client application in response to a registration request from the Java client application. See Spender, column 4, lines 12-21. Further, to service requests from the Java client applications, Spender discloses that the manager and dispatcher applications are written in a native C system language and they communicate with the Java client applications via a Java Native Interface (JNI). See Spender, column 4, lines 17-31 and 38-46. Spender is silent on the content of the requests from the Java client applications.

However, Spender at least fails to disclose a Java Native Interface to obtain the status of Java processes and to update shared memory with the obtained status. Moreover, Spender fails to disclose that the status of Java processes is stored in shared memory.

D. Rejection of Claims 1-3, 6, 7, 9, 10, 13, 14, 16, 17, 19-23, 25, 26, and 28-39 Under 35 U.S.C. § 103

Claims 1-3, 6, 7, 9, 10, 13, 14, 16, 17, 19-23, 25, 26, and 28-39 stand rejected under 35 U.S.C. § 103(a) as being unpatentable over U.S. Patent Publication No. 2005/0005200 issued to Matena et al. (hereinafter "Matena") in view of U.S. Patent No. 5,991,893 issued to Snider (hereinafter "Snider").

To establish a *prima facie* case of obviousness the Examiner must set forth a clear articulation of the reasons that the claimed invention would have been obvious. The reasoning cannot be based on mere conclusory statements. See KSR Int'l Co. v. Teleflex Inc. (KSR), 82 USPQ2d 1385, 1396 (2007) and MPEP § 2142. Further, the Federal Circuit has clarified that the determination of the proper combination of prior art teachings in light of the Supreme Court's decision in KSR Int'l Co. v. Teleflex Inc. is to be based on the flexible application of the teaching, suggestion and motivation (TSM) test, because "as the Supreme Court suggests, a flexible approach to the TSM test prevents hindsight and focuses on evidence before the time of the invention." In re Translogic Tech., Inc., 504 F.3d 1249, 1257 (Fed. Cir. 2007).

However, as discussed below, the cited references fail to teach or suggest each element of claims 1-3, 6, 7, 9, 10, 13, 14, 16, 17, 19-23, 25, 26, and 28-39 and the Examiner has failed to establish any teaching, suggestion or motivation to combine the cited references.

1. **Claims 1, 2, 6, 7, 9, 10, 13, 14, 16, 17, 19, 20, 21, 23, 25, 26, 28, 29, and 33**

a) Independent Claim 1 Is not Obvious at Least Because Matena in View of Snider Fails to Teach or Suggest a Communication Interface Coupled Between the Launch Logic and the Control Logic.

Claim 1 recites the elements of "a communication interface coupled between the launch logic and the control logic." Matena fails to teach or suggest these elements. In the Final Office Action, the Examiner has pointed to paragraphs [0412] and [0413] of Matena as allegedly teaching or suggesting these elements. However, Appellant submits that the Examiner has failed to sufficiently articulate the rationale for establishing obviousness of these elements as required under KSR. Under KSR, the Examiner is required to set forth a clear articulation of the reasons that the claimed invention would have been obvious and the reasoning cannot be based on mere conclusory statements. See KSR Int'l Co. v. Teleflex Inc. (KSR), 82 USPQ2d 1385, 1396 (2007) and MPEP § 2142. In the Final Office Action, the Examiner has failed to articulate how these portions of Matena teach or suggest "a communication interface" that satisfies the interrelationship of being "coupled between the launch logic and the control logic." Therefore, by failing to provide the reasoning for relying on these portions of Matena, the Examiner has failed to meet the requirement under KSR to articulate a rationale for showing a *prima facie* case of obviousness of these elements.

In addition, notwithstanding the Examiner's omission, a plain reading of Matena demonstrates that not every element of the claim 1 is expressly taught or suggested. The sections of Matena cited by the Examiner disclose that the node controllers are implemented using the Java 2 platform, or in an alternate embodiment, implemented using a native programming language (e.g., C++). See Matena, paragraphs [0412] and [0413]. However, there is no express indication in these sections of Matena that disclose "a communication interface" having the interrelationship of "coupled between the launch logic and the control logic." Although, communication between nodes is described as being accomplished using a RMI API or via shared memory, Matena is silent in regard to the location of either the RMI API or the shared memory, let alone,

that a communication interface is "coupled between the launch logic and the control logic." Id.

Further, the Examiner has not established that these limitations are inherently taught or suggested by the reference. To establish inherency, the descriptive matter must *necessarily* be present in the cited reference.

> The fact that a certain result or characteristic may occur or be present in the prior art is not sufficient to establish the inherency of that result or characteristic. *In re Rijckaert*, 9 F.3d 1531, 1534, 28 USPQ2d 1955, 1957 (Fed. Cir. 1993) (reversed rejection because inherency was based on what would result due to optimization of conditions, not what was necessarily present in the prior art); *In re Oelrich*, 666 F.2d 578, 581-82, 212 USPQ 323, 326 (CCPA 1981). "To establish inherency, the extrinsic evidence 'must make clear that the missing descriptive matter is necessarily present in the thing described in the reference, and that it would be so recognized by persons of ordinary skill. Inherency, however, may not be established by probabilities or possibilities. The mere fact that a certain thing may result from a given set of circumstances is not sufficient.' " *In re Robertson*, 169 F.3d 743, 745, 49 USPQ2d 1949, 1950-51 (Fed. Cir. 1999) (citations omitted) (The claims were drawn to a disposable diaper having three fastening elements. The reference disclosed two fastening elements that could perform the same function as the three fastening elements in the claims. The court construed the claims to require three separate elements and held that the reference did not disclose a separate third fastening element, either expressly or inherently.)

See MPEP § 2112(IV) (emphasis in original). The aspect of a communication interface coupled between the launch logic and control logic is not *necessarily* present in Matena.

For example, again, Matena indicates that communication between nodes may be accomplished using Java RMI or shared memory. See Matena, paragraphs [0412] and [0413]. The Java RMI API is *a software component* provided by a Java virtual machine on a respective node. However, the location of the RMI API is at a location determined by the Java virtual machine rather than being *necessarily* coupled between the launch logic and control logic. Moreover, shared memory is *a hardware component* and may be located in a respective node (i.e., on the system logic board) instead of being *necessarily* coupled between the launch logic and control logic. Thus, the Examiner has not

established that Matena expressly or inherently teaches or suggest the elements of "a communication interface coupled between the launch logic and the control logic."

Further, Snider fails to teach or suggest the above missing elements. The Examiner has not cited and Appellant is unable to discern the portion of Snider that teaches or suggest the above missing elements. Thus, for at least the previous reasons, Matena in view of Snider fails to teach or suggest each element of claim 1. Therefore, the Examiner has failed to establish a *prima facie* case of obviousness.

Consequently, in view of at least these reasons, Matena in view of Snider fails to teach or suggest each element in claim 1. Thus, in view of at least the foregoing reasons, claim 1 is directed toward allowable subject matter. Accordingly, Appellant respectfully requests that the § 103 rejection of claim 1 be overturned.

b) Independent Claim 1 Is not Obvious at Least Because Matena in View of Snider Fails to Teach or Suggest a Control Logic that Accesses Shared Memory to Obtain Status of Java Processes.

Claim 1 recites the elements of "a communication interface . . . to enable the launch logic to obtain status of each of the Java processes and enable the control logic to access the status in a shared memory via the communication interface." Matena fails to teach or suggest these elements. As discussed above, the portion of Matena cited by the Examiner as allegedly teaching or suggesting these elements discloses that communication between nodes may be implemented using a Java RMI API and, alternatively, the nodes may communicate with each other via shared memory. See Matena, paragraphs [0412] and [0413].

In the Final Office Action, the Examiner has cited to these sections of Matena without offering an explanation as to how Matena allegedly teaches or suggests these elements. Under KSR, a *prima facie* case of obviousness cannot be established by mere conclusory statements and there must be a clear articulation of the reasons that the claimed invention would have been obvious based on the prior art. See KSR Int'l Co. v. Teleflex Inc. (KSR), 82 USPQ2d 1385, 1396 (2007) and MPEP § 2142. Because the Examiner has failed to articulate a rationale for supporting the rejection based on these

portions of Matena, the rejection fails to be sufficient to show a *prima facie* case of obviousness of the claimed elements under KSR.

Moreover, a plain reading of Matena illustrates that these elements are not expressly taught or suggested in Matena. For example, paragraphs [0412] and [0413] of Matena fail to disclose the *content and type of communication* that is performed between the nodes, let alone, communication "to obtain status of each of the Java processes and . . . to access the status in a shared memory," as recited in claim 1. Instead, Matena discloses that a node may provide an API for subscribing to the node's event notifications. See Matena, paragraph [0107]. In particular, the node sends an event notification to event subscribers upon detection of a process failure. See Matena, paragraph [0105]. Therefore, the event notification does not include the status of *each of the processes* because only an indication of that *particular failed process* is present. Consequently, in view of at least these reasons, Matena fails to expressly teach or suggest each element in claim 1.

Further, the Examiner has not established that these limitations are inherently taught or suggested by the reference. To establish inherency, the descriptive matter must *necessarily* be present in the cited reference. See MPEP § 2112(IV). The aspect of a communication interface to obtain status of each of the Java processes and to access the status in a shared memory is not *necessarily* present in Matena. For example, again, Matena discloses that event notifications related to a failed process are sent upon detection of the failed process. See Matena, paragraphs [0105] and [0107]. However, Matena discloses that the event notification is in the form of *a text-based message* sent over TCP/IP rather then being "shared memory" as recited in claim 1. See Matena, paragraph [0109]. Moreover, the event notification only relates to the failed process instead of each of the processes. Thus, in view of at least these reasons, the Examiner has not established that Matena expressly or inherently teaches or suggest "a communication interface . . . to enable the launch logic to obtain status of each of the Java processes and enable the control logic to access the status in a shared memory via the communication interface."

In addition, Snider fails to teach or suggest the above missing elements. The Examiner has not cited and Appellant is unable to discern the portion of Snider that

teaches or suggest the above missing elements. Thus, for at least the previous reasons, Matena in view of Snider fails to teach or suggest each element of claim 1. Therefore, the Examiner has failed to establish a *prima facie* case of obviousness.

Thus, in view of at least the foregoing reasons, claim 1 is directed toward allowable subject matter. Accordingly, Appellant respectfully requests that the § 103 rejection of claim 1 be overturned.

c) Claim 1 Is not Obvious at Least Because Matena in View of Snider Fails to Teach or Suggest a Launch Logic to Store and Maintain Status in Shared Memory Via a Communication Interface.

Appellant submits the following reasons in addition to the reasons discussed above to further establish the patentability of claim 1.

Claim 1 recites the additional elements of "the launch logic to store and maintain the status in the shared memory via the communication interface." The Examiner on page 3 of the Final Office Action admitted that Matena fails to teach or suggest these elements. The Examiner then relied upon Snider as allegedly teaching or suggesting these missing elements. However, a plain reading of Snider fails to show that Snider expressly teaches or suggests these missing elements.

The portion of Snider cited by the Examiner discloses a function implemented in the C programming language that (when called by the operating system) allocates a data structure in the 'virtually reliable memory.' See Snider, column 5, lines 32-35. The virtually reliable memory (i.e., "VRSM") is disclosed as *a software layer* that represents the underlying hardware as a single resource to the operating system (OS) and that also allocates data structures for use by the operating system. See Snider, column 4, lines 62-65; column 5, lines 2-5; Fig. 1. However, Snider fails to teach or suggest that the OS or the software layer performs the elements of "store and maintain the status in the shared memory," as recited in claim 1. As required in MPEP § 2143.03, all words in a claim must be considered in judging the patentability of that claim against the prior art. See In re Wilson, 424 F.2d 1382, 1385, 165 USPQ 494, 496 (CCPA 1970). In claim 1, the

"status" relates to "each of the Java processes." Although Snider discloses that the data structures contain "critical information" that allows the system to recover from a failure, the critical information is not disclosed as being related to a "status of each of the Java processes," as recited in claim 1. See Snider, column 5, lines 5-7. In other words, Snider fails to *expressly teach or suggest* that the critical information includes the "status of each of the Java processes." Moreover, in the context of claim 1, the "launch logic" is recited as executing "Java processes in each respective server node." Snider, however, is silent in regard to whether the OS or the software layer performs the tasks of executing Java processes in each node. Appellant also notes that neither the OS nor the software layer is equivalent to "the launch logic" because they do not perform the elements of obtaining "status of each of the Java processes." Consequently, for at least these reasons, Snider fails to expressly teach or suggest the elements of "the launch logic to store and maintain the status in the shared memory via the communication interface."

Further, the Examiner has not established that these limitations are inherently taught or suggested by the reference. To establish inherency, the descriptive matter must *necessarily* be present in the cited reference. See MPEP § 2112(IV). The aspect of "the launch logic to store and maintain the status in the shared memory via the communication interface" is not *necessarily* present in Snider. For example, again, Snider discloses that critical information is included in the data structure. See Snider, column 5, lines 5-7. Snider then discloses that information included in the data structure may include a lock field, a pointer to replicated information, and a checksum. See Snider, column 6, lines 40-56. However, none of previous information in the data structure indicates that "the status" (i.e., status of each of the Java processes) is *necessarily* present in the data structure.

Thus, in view of at least these reasons, the Examiner has not established that Snider expressly or inherently teaches or suggest "the launch logic to store and maintain the status in the shared memory via the communication interface," as recited in claim 1. Therefore, in view of at least the foregoing reasons, Matena in view of Snider fails to teach or suggest each element of claim 1. As a result, the Examiner has failed to establish a *prima facie* case of obviousness.

Thus, in view of at least the foregoing additional reasons, claim 1 is separately patentable over the art of record. Accordingly, Appellant respectfully requests that the § 103 rejection of claim 1 be overturned.

d) Claim 1 Is not Obvious at Least Because Snider Changes Matena's Principle of Operation.

Snider, in addition, changes Matena's principle of operation. MPEP § 2143.01(VI) states that if the proposed modification or combination of the references would change the principle of operation of the prior art invention being modified, then the references are not sufficient to render the claims *prima facie* obvious. See also In re Ratti, 123 USPQ 349 (CCPA 1959). In particular, Matena discloses that one advantage of a distributed computing system over a single computer system is that performance can be increased by adding nodes (e.g., more computer systems) connected over a network (e.g., servers connected by Ethernet). See Matena, paragraphs [0006] and [0097]. Matena further discloses that the execution control system includes distributed computing systems with nodes on *different computer architectures* and nodes that use *different operating systems*. See Matena, paragraph [0078], lines 30-33. Snider, in contrast, discloses *a single operating system* that relies upon a software layer to abstract the underlying hardware related to the nodes in the system. See Snider, Fig. 1; column 3, line 67 to column 4, line 8; column 4, lines 53-56. A *single operating system* and its underlying hardware run on a *single computer architecture* instead of different computer architectures and operating systems as disclosed in Matena. The Examiner stated (see page 3 of the Final Office Action) that the skill artisan would be motivated to combine the cited references "by the improvement in performance offered by using shared memory between the nodes." However, Snider's teaching of shared memory is reliant on a single operating system running on a single computer architecture and would not be applicable to nodes with different computer architectures and operating systems. Thus, Snider's shared memory system cannot be combined with Matena's system, because it would require changing Matena's principle of operation of including nodes of different architectures and operating systems.

Hence, for at least the reasons set forth above, there would be no motivation to combine Matena and Snider because Snider would change Matena's principle of

operation. Thus, the Examiner has failed to establish a *prima facie* case of obviousness. Therefore, claim 1 is directed to patentable subject matter for these additional reasons. Accordingly, Appellant respectfully requests that the § 103 rejection of claim 1 be overturned.

e) Independent Claims 13, 17, 21, and 28 Recite Analogous Elements to Those in Claim 1.

Independent claims 13, 17, 21, and 28 recite analogous elements to those in claim 1. Thus, for at least the arguments made above in connection with claim 1, claims 13, 17, 21, and 28 are patentable over Matena in view of Snider because the cited art fails to teach or suggest each element of these claims. Therefore, for at least the reasons set forth above, the Examiner has failed to establish a *prima facie* case of obviousness for claims 13, 17, 21, and 28.

Thus, in view of at least the above reasons, claims 13, 17, 21, and 28 are directed to patentable subject matter. Accordingly, Appellant respectfully requests that the § 103(a) rejection of claims 13, 17, 21, and 28 be overturned.

f) Claims 2, 6, 7, 9, 10, 14, 16, 19, 20, 22, 23, 25, 26, 29, 30, 32 and 33 Depend on Respective Patentable Base Claims.

Dependent claims 2, 6, 7, 9, and 10 depend on claim 1 and incorporate the limitations thereof. Dependent claims 14 and 16 depend on claim 13 and incorporate the limitations thereof. Dependent claims 19 and 20 depend on claim 17 and incorporate the limitations thereof. Dependent claims 22, 23, 25, and 26 depend on claim 21 and incorporate the limitations thereof. Dependent claims 29, 30, 32, and 33 depend on claim 28 and incorporate the limitations thereof.

Thus, for at least the reasons discussed previously in connection with the respective base claims of claims 2, 6, 7, 9, 10, 14, 16, 19, 20, 22, 23, 25, 26, 29, 30, 32 and 33, Matena in view of Snider fails to teach or suggest each element of claims 2, 6, 7, 9, 10, 14, 16, 19, 20, 23, 25, 26, 29, 30, 32 and 33. Therefore, claims 2, 6, 7, 9, 10, 14, 16, 19, 20, 22, 23, 25, 26, 29, 30, 32 and 33 are patentable over the art of record because each of these claims depends on either claim 1, 13, 17, 21, or 28.

Thus, in view of at least the foregoing reasons, claims 2, 6, 7, 9, 10, 14, 16, 19, 20, 22, 23, 25, 26, 29, 30, 32 and 33 are directed toward allowable subject matter. Accordingly, Appellant respectfully requests that the § 103 rejection of claims 2, 6, 7, 9, 10, 14, 16, 19, 20, 22, 23, 25, 26, 29, 30, 32 and 33 be overturned.

2. Claim 3

a) Claim 3 Is at Least Patentable Over Matena in View of Snider Because the Cited Art Fails to Teach or Suggest a Shared Memory to Store Status of Java Processes.

Claim 3 recites the elements of "the shared memory to store the status of the Java processes." The Examiner has cited a section of Matena as teaching or suggesting these elements. In contrast, again, the portion of Matena relied upon by the Examiner as allegedly disclosing these elements, instead, discloses that nodes may communicate with each other using shared memory. See Matena, paragraph [0413]. The Examiner further stated that "[s]ince shared memory can be used to send the status messages to other nodes, those status messages must be stored in shared memory." However, this portion of Matena fails to expressly teach or suggest the content of the communication between the nodes, let alone, that the communication would require storing status messages in shared memory as alleged by the Examiner.

Moreover, the Examiner has failed to establish that Matena inherently teaches or suggest these elements. To establish inherency, the descriptive matter must *necessarily* be present in the cited reference. See MPEP § 2112(IV). However, the aspect of "the shared memory to store the status of the Java processes" is not *necessarily* present in Matena. In particular, communication between nodes, instead, could involve a transient transmission of messages that do not require storage in shared memory. For example, a node could send a ping request to another node and then wait for ping response. Because a ping request and response involves timing data (i.e., the round trip time), the nodes communicate directly with each other instead of relying on shared memory.

Further, consistent with the above discussion, Matena discloses that event notifications related to a failed process are sent upon detection of the failed process. See

Matena, paragraphs [0105] and [0107]. However, Matena discloses that the event notification is in the form of *a text-based message* sent over TCP/IP rather then being "shared memory" as recited in claim 1. See Matena, paragraph [0109]. In addition, the event notification only relates to a single failed process instead of each of the processes. Therefore, the Examiner has failed to demonstrate that shared memory sends status messages and status messages are stored in shared memory are *necessarily* present in Matena. Consequently, for at least the previous reasons, Matena fails to expressly or inherently teach or suggest each element of claim 3.

Moreover, based on the portion of Matena cited by the Examiner, Appellant submits that the Examiner has improperly relied on impermissible hindsight to match paragraph [0413] of Matena with the elements in claim 3. MPEP § 2142 states that in establishing a *prima facie* case of obviousness, impermissible hindsight must be avoided and the legal conclusion must be reached on the basis of the facts gleaned from the prior art. However, the Examiner has failed to cite other sections of Matena to support the assertion that shared memory is used to send the status messages and, as a result, status messages are stored in shared memory. Therefore, without first scanning Appellant's Specification, the Examiner would be unable to reach the conclusion based on the portion of Matena cited by the Examiner.

Further, the Examiner has failed to cite and Appellant is unable to discern the portion of Snider that allegedly teaches or suggests the missing elements. Thus, in view of at least the foregoing reasons, Matena in view of Snider fails to teach or suggest each element of claim 3. Therefore, the Examiner has failed to establish a *prima facie* case of obviousness.

Thus, in view of at least the reasons set forth above, claim 3 is separately patentable over the cited art. Accordingly, Appellant respectfully requests that the § 103 rejection of claim 3 be overturned.

b) Claim 3 Depends on Patentable Base Claim 1.

Claim 3 depends on claim 1 and incorporates the limitations thereof. Thus, for at least the reasons discussed above in connection with claim 1, Matena in view of Snider

fails to teach or suggest each element of claim 3. Therefore, claim 3 is patentable over the art of record because claim 3 depends on claim 1.

Thus, in view of at least the foregoing reasons, claim 3 is directed toward allowable subject matter. Accordingly, Appellant respectfully requests that the § 103 rejection of claim 3 be overturned.

3. Claim 31

a) Claim 31 Is at Least Patentable Over Matena in View of Snider Because the Cited Art Fails to Teach or Suggest a Means for Obtaining Status and Updating Shared Memory with the Obtained Status.

Claim 31 recites the elements of "means for obtaining status for each of the Java processes," and "means for updating the shared memory with the obtained status." Because claim 31 recites "means for" claim language, the Examiner must interpret the means plus function claim language to read on *only the structures or materials disclosed in the specification and equivalents thereof* (including the manner in which the claimed functions are performed) according to 35 U.S.C. § 112, sixth paragraph. See In re Donaldson Co., 16 F.3d 1189 at 1194 (Fed. Cir. 1994); MPEP § 2181(I); MPEP § 2106. With respect to interpreting the elements of a "means for obtaining status" and a "means for updating the share memory" under § 112, sixth paragraph, as disclosed in paragraph [00018] of the Specification, the status information is obtained by using a Java native interface (JNI) implemented within the launch logic. Moreover, the JNI is invoked by the launch logic to update the shared memory with information relating to the status of the Java processes. See Specification, paragraph [00018]. Therefore, in light of the Specification, a "means for obtaining status" and "a means for updating the shared memory" should be interpreted as the JNI implemented within the launch logic.

The Examiner has cited a section of Matena as teaching or suggesting these elements. However, Matena fails to teach or suggest the elements of "means for obtaining status" and "a means for updating the shared memory" based on an interpretation in light of the Specification. The section of Matena cited by the Examiner, instead, discloses that the Java application controller (JAC) may implement an API to

expose its operations (e.g., the respond to hardware changes and obtain application information operations). See Matena, paragraphs [0214] and [0221]. However, Matena discloses that the operations provided in the API are called by other programs (e.g., a system management tool and other components) rather than called by the Java application controller itself. See Matena, paragraph [0221]. In addition, Matena is silent on whether the API is implemented as a JNI as disclosed in the Specification. Moreover, again, the Examiner cited the portion of Matena that discloses that the nodes may communicate using shared memory. See Matena, paragraph [0413]. However, this portion of Matena also fails to disclose that a JNI is used to update shared memory as disclosed in the Specification, let alone, that the JNI is implemented within the Java application controller. Thus, for at least these reasons, Matena fails to teach or suggest the elements related to a "means for obtaining status" and "a means for updating the shared memory" as interpreted in light of the Specification.

Further, the Examiner has failed to cite and Appellant is unable to discern the portion of Snider that teaches or suggests the above missing elements. Consequently, in view of at least the foregoing reasons, Matena in view of Snider fails to teach or suggest the elements of "means for obtaining status for each of the Java processes," and "means for updating the shared memory with the obtained status," as recited in claim 31. Therefore, the Examiner has failed to establish a *prima facie* case of obviousness.

Thus, for at least the reasons set forth above, claim 31 is separately patentable over Matena in view of Snider. Accordingly, Appellant respectfully requests that the § 103(a) rejection of claim 31 be overturned.

b) Claim 31 Depends on Patentable Base Claim 28.

Claim 31 depends on claim 28 and incorporates the limitations thereof. Thus, for at least the reasons discussed above in connection with claim 28, Matena in view of Snider fails to teach or suggest each element of claim 31. Therefore, claim 31 is patentable over the art of record because claim 31 depends on claim 28.

Thus, in view of at least the foregoing reasons, claim 31 is directed toward allowable subject matter. Accordingly, Appellant respectfully requests that the § 103 rejection of claim 31 be overturned.

4. Claims 34, 36, 37, 38, and 39

a) Claim 34 Is at Least Patentable Over Matena in View of Snider Because the Cited Art Fails to Teach or Suggest the Launch Logic Stores the Status Independent of the Control Logic Accessing the Status.

Claim 34 recites the elements of "the launch logic stores the status independent of the control logic accessing the status." In the Final Office Action, the Examiner has failed to cite and Appellant is unable to discern the portion of Matena that allegedly teaches or suggests the missing elements. The Examiner, instead, cited portions of Snider as allegedly disclosing the above elements (see page 11 of the Final Office Action). However, Snider discloses a procedure for locking shared memory during a write to the shared memory. See Snider, column 6, lines 64-67. Because independent writes and reads (e.g., when more than 34 processor reads and write data) to the shared memory would result in corruption of data, Snider locks access to the shared memory to preserve the data's integrity. See Snider, column 7, lines 34-7. Therefore, Snider's locking mechanism prevents an independent exchange of data because the processors are required to serially access the data. See Snider, column 7, lines 7-9. Thus, upon obtaining the lock by the processor, access to the data by another processor is precluded until the lock is released by the processor. In other words, Snider discloses that storing and accessing the data is *dependent upon the processor obtaining the lock*. Further, no other processor may attempt to write or access the data until the lock is relinquished. Consequently, Snider fails to disclose that each processor can independently access and store data to the shared memory. Thus, for at least these reasons, Snider fails to teach or suggest the elements of "the launch logic stores the status *independent of the control logic accessing the status*," (emphasis added) as recited in claim 34.

Further, the Examiner has misinterpreted the meaning of the term "independent" as recited in claim 34. As required by the MPEP, the words of a claim

must be given their plain meaning by referring to the ordinary and customary meaning given to the term by those of ordinary skill in the art. See MPEP, § 2111.01(I) and § 2111.01(III). Moreover, the ordinary and customary meaning of a term may be evidenced by a variety of sources, including extrinsic evidence. See Phillips v. AWH Corp., 415 F.3d at 1314. The term "independent" is commonly defined as "not depending or contingent upon something else for existence, operation, etc." "independent." Dictionary.com. 23 April 2008. <http://dictionary.reference.com/search?q=independent>. However, in the Final Office Action, the Examiner stated that "[t]he system [in Snider] automatically allows for nodes to access the shared memory independently." In contrast, as discussed above, access to the shared memory is *dependent on the processor acquiring the lock* that denies another processor from writing and accessing the data while the lock is held. Obtaining the lock is a dependency that must be satisfied before permitting access to the shared memory. Other processors are then denied access to the shared memory. In light of the common usage of the term "independent," relying on a lock for access to the shared memory should not be considered an independent accessing and storing of the shared memory. Thus, the Examiner has confused the procedure of locking shared memory with independently accessing and storing to the shared memory. As a result, in view of the common definition of the term "independent," the Examiner has failed to identify the section of Snider that discloses the elements of "the launch logic stores the status *independent of the control logic accessing the status*," (emphasis added) as recited in claim 34.

Thus, in view of at least the foregoing reasons, Matena in view of Snider fails to teach or suggest each element of claim 34. Therefore, the Examiner has failed to establish a *prima facie* case of obviousness.

In view of at least the foregoing reasons, claim 34 is separately patentable over the art of record. Accordingly, Appellant respectfully requests that the § 103 rejection of claim 34 be overturned.

b) Claim 34 Depends on Patentable Base Claim 1.

In addition, claim 34 depends on claim 1 and incorporates the limitations thereof. Thus, for at least the reasons discussed above in connection with claim 1, Matena in view of Snider fails to teach or suggest each element of claim 34. Therefore, claim 34 is patentable over the art of record because claim 34 depends on claim 1.

Thus, in view of at least the foregoing reasons, claim 34 is directed toward allowable subject matter. Accordingly, Appellant respectfully requests that the § 103 rejection of claim 34 be overturned.

c) Claims 36, 37, 38, and 39 Recite Analogous Elements to Those Recited in Claim 34.

Claims 36-39 recite analogous elements to those in claim 34. Thus, for at least the reasons discussed in connection with claim 34, claims 36-39 are patentable over Matena in view of Snider because the cited art fails to teach or suggest each element of these claims. Therefore, for at least the reasons set forth above, the Examiner has failed to establish a *prima facie* case of obviousness for claims 36-39.

Thus, in view of at least the previous reasons, claims 36-39 are directed to patentable subject matter. Accordingly, Appellant respectfully requests that the § 103(a) rejection of claims 36-39 be overturned.

d) Claims 36, 37, 38, and 39 Depend on Patentable Base Claims.

Dependent claims 36, 37, 38, and 39 depend on base claims 13, 17, 21, and 28, respectively, and incorporate the limitations thereof. Thus, for at least the reasons discussed previously in connection with the respective base claims of claims of 36, 37, 38, and 39, Matena in view of Snider fails to teach or suggest each element of claims 36, 37, 38, and 39. Therefore, claims 36, 37, 38, and 39 are patentable over the art of record because each of these claims depends on either claim 1, 13, 17, 21, or 28.

Thus, in view of at least the foregoing reasons, claims 36, 37, 38, and 39 are directed toward patentable subject matter. Accordingly, Appellant respectfully requests that the § 103 rejection of claims 36, 37, 38, and 39 be overturned.

5. **Claim 35**

a) Claim 35 Is Patentable Over Matena in View of Snider Because the Cited Art Fails to Teach or Suggest a Persistent Data Structure Is Stored in the Shared Memory to Enable an Independent Exchange of Information Between the Java Processes.

Claim 35 recites the elements of "a persistent data structure is stored in the shared memory to enable an independent exchange of information between the Java processes." The Examiner has not cited and Appellant is unable to discern the portion of Matena that allegedly teaches or suggest the above elements. Instead, the Examiner has cited Snider to teach or suggest these elements. However, as discussed previously in connection with claim 35, the Examiner has cited the section of Snider that discloses a procedure for locking shared memory during a write to the shared memory. See Snider, column 6, lines 64-67. Because Snider locks access to the shared memory, an independent exchange of information stored in the shared memory is not achieved. See Snider, column 7, lines 1-7. Instead, the processors serially access the data that requires each processor to wait for the lock to be released and then acquire the lock. See Snider, column 7, lines 5-9. Therefore, Snider discloses that an exchange of the data in the shared memory is *dependent upon the processor obtaining the lock to the exclusion of another processor*. As a result, Snider fails to teach or suggest that the shared memory enables an independent exchange of information in the shared memory. Moreover, Appellant notes that Snider fails to disclose Java processes as recited in claim 35. Thus, for at least these reasons, Snider fails to teach or suggest the elements of "a persistent data structure is stored in the shared memory to enable an independent exchange of information between the Java processes," as recited in claim 35.

Thus, in view of at least the foregoing reasons, Matena in view of Snider fails to teach or suggest each element of claim 35. Therefore, the Examiner has failed to establish a *prima facie* case of obviousness.

In view of at least the foregoing reasons, claim 35 is separately patentable over the art of record. Accordingly, Appellant respectfully requests that the § 103 rejection of claim 35 be overturned.

b) Claim 35 Depends on Patentable Base Claim 1.

Moreover, claim 35 depends on claim 1 and incorporates the limitations thereof. Thus, for at least the reasons discussed above in connection with claim 1, Matena in view of Snider fails to teach or suggest each element of claim 35. Therefore, claim 35 is patentable over the art of record because claim 35 depends on claim 1.

Thus, in view of at least the foregoing reasons, claim 35 is directed toward allowable subject matter. Accordingly, Appellant respectfully requests that the § 103 rejection of claim 35 be overturned.

E. Rejection of Claims 4, 5, 8, 11, 12, 15, 18, 24, and 27 Under 35 U.S.C. § 103(a)

Claims 4, 5, 8, 11, 12, 15, 18, 24, and 27 stand rejected under 35 U.S.C. § 103(a) as being obvious over Matena in view Snider in further view of U.S. Patent No. 6,823,358 issued to Spender (hereinafter "Spender").

However, as discussed below, Matena in view Snider in further view of Spender fails to teach or suggest each element of claims 4, 5, 8, 11, 12, 15, 18, 24, and 27

1. Claims 4, 18, and 24

a) Claim 4 is Patentable at Least Because Matena in View of Snider in Further View of Spender Fails to Teach or Suggest a Java Native Interface to Obtain the Status of Java Processes.

Claim 4 recites the elements of "a Java native interface to obtain the status of each of the Java processes and to update the shared memory with the obtained status." The Examiner conceded that Matena fails to teach or suggest these elements (see pages 13 and 14 of the Final Office Action). The Examiner then stated that "since it is well known that Java does not allow for access [*sic*] native system resources . . . such as shared memory and named pipes without the use of the Java native interface, then it is

impliedly taught by Matena that the Java native interface is used to access the shared memory used for communication." Appellant interprets the Examiner's statement as meaning that Matena inherently discloses the use of a Java native interface (JNI) to access the shared memory. To establish inherency, the extrinsic evidence must make clear that the missing descriptive matter is *necessarily present* in the thing described in the reference, and that it would be so recognized by persons of ordinary skill. See In re Robertson, 49 USPQ2d 1949, 1950-51 (Fed. Cir. 1999). However, as discussed below, the use of a JNI is not necessarily present to access the shared memory as alleged by the Examiner.

In contrast, the portion of Matena cited by the Examiner, again, discloses that nodes may communicate using shared memory. See Matena, paragraph [0413]. Further, this portion of Matena discloses that the execution controllers, application controllers, and node controllers on the nodes may be implemented using native system programming languages (e.g., C, C++, C#). Id. Natively implemented applications can then access platform-specific resources by directly calling native methods without the additional layer of the JNI. Therefore, *by calling native methods*, the natively implemented execution controllers, application controllers, and node controllers could communicate using shared memory without requiring the use of JNI. Thus, the use of a JNI to access shared memory is not necessarily present in the system as alleged by the Examiner.

Further, the Examiner has failed to cite and Appellant is unable to discern the portion of Snider that teaches or suggests the missing elements. Instead, the Examiner has relied upon Spender to disclose the missing elements of "a Java native interface to obtain the status of each of the Java processes and to update the shared memory with the obtained status," as recited in claim 4. However, Spender fails to teach or suggest these missing elements. In the Final Office Action (see page 14), the Examiner alleged that Spender teaches a JNI to access shared memory. In contrast, Spender discloses that Java applications may use a JNI to call methods related to registering and sending requests to a manager and dispatcher application, respectively. See Spender, column 8, lines 12-25 and 38-44. Further, Spender discloses that the natively implemented DLL may communicate with the manager and dispatcher using shared memory. However,

the DLL provides operations not related to obtaining "status of Java processes" and updating "shared memory with the obtained status" but for responding to requests from the Java application. See Spender, column 4, lines 21-31. Spender also fails to disclose the content of these requests from the Java application, let alone, that the requests relate to the "status of Java processes," as recited in claim 4. As a result, for at least the previous reasons, Spender fails to teach or suggest each element of claim 4.

Thus, in view of at least the foregoing reasons, Matena in view of Snider in further view of Spender fails to teach or suggest each element of claim 4. Therefore, the Examiner has failed to establish a *prima facie* case of obviousness.

In view of at least the foregoing reasons, claim 4 is separately patentable over the art of record. Accordingly, Appellant respectfully requests that the § 103 rejection of claim 4 be overturned.

b) Claim 4 Depends on Patentable Base Claim 1.

In addition, claim 4 depends on base claim 1 and incorporates the limitations thereof. Thus, for at least the reasons discussed above in connection with claim 1, Matena in view of Snider fails to teach or suggest each element of claim 4. Further, the Examiner has not cited and Appellant is unable to discern the portion of Spender that allegedly disclose the missing elements related to "a communication interface . . . to enable . . . the control logic to access the status in a shared memory via the communication interface, the launch logic to store and maintain the status in the shared memory via the communication interface," as recited in claim 1. Consequently, for at least these reasons, Matena in view of Snider in further view of Spender fails to teach or suggest each element of claim 1. Therefore, claim 4 is patentable over the art of record because claim 4 depends on claim 1.

Thus, in view of at least the foregoing reasons, claim 4 is directed toward allowable subject matter. Accordingly, Appellant respectfully requests that the § 103 rejection of claim 4 be overturned.

c) Claims 18 and 24 Recite Analogous Limitations to Those in Claim 4.

Claims 18 and 24 recite analogous elements to those in claim 4. Thus, for at least the reasons discussed in connection with claim 4, claims 18 and 24 are patentable over Matena in view of Snider in further view of Spender because the cited art fails to teach or suggest each element of these claims. Therefore, for at least the reasons set forth above, the Examiner has failed to establish a *prima facie* case of obviousness for claims 18 and 24.

Thus, in view of at least the previous reasons, claims 18 and 24 are directed to patentable subject matter. Accordingly, Appellant respectfully requests that the § 103(a) rejection of claims 18 and 24 be overturned.

a) Claims 18 and 24 Depend on Patentable Base Claims.

Claims 18 and 24 depend on base claims 17 and 21, respectively, and incorporate the limitations thereof. Thus, for at least the reasons discussed above in connection with claims 17 and 21, Matena in view of Snider fails to teach or suggest each element of claim 35. In addition, the Examiner has not cited and Appellant is unable to discern the section of Spender that allegedly teaches or suggest the missing elements. Consequently, in view of at least the foregoing reasons, Matena in view of Snider in further view of Spender fails to teach or suggest each element of claims 18 and 24 because these claims either depend on claim 17 or 21.

Thus, in view of at least the foregoing reasons, claims 18 and 24 are directed toward allowable subject matter. Accordingly, Appellant respectfully requests that the § 103 rejection of claims 18 and 24 be overturned.

2. Claims 5, 8, 11, 12, 15, and 27

a) Claims 5, 8, 11, 12, 15, and 27 Depend on Patentable Base Claims.

Dependent claims 5, 8, 11, 12, 15, and 27 depend on base claims 1, 13, 17, or 21, respectively, and incorporate the limitations thereof. Thus, for at least the reasons

discussed previously in connection with the respective base claims of claims of 5, 8, 11, 12, 15, and 27, Matena in view of Snider fails to teach or suggest each element of claims 5, 8, 11, 12, 15, and 27. Further, the Examiner has not cited and Appellant is unable to discern the portion of Spender that discloses the missing elements. Therefore, claims 5, 8, 11, 12, 15, and 27 are patentable over the art of record because each of these claims depends on either base claim 1, 13, 17, or 21.

Thus, in view of at least the foregoing reasons, claims 5, 8, 11, 12, 15, and 27 are directed toward patentable subject matter. Accordingly, Appellant respectfully requests that the § 103 rejection of claims 5, 8, 11, 12, 15, and 27 be overturned.

For the reasons set forth above, the Appellant respectfully requests the Board overturn the rejection of claims 1-39 as being obvious in view of the art of record.

Respectfully submitted,
BLAKELY, SOKOLOFF, TAYLOR & ZAFMAN LLP

Date: April 30, 2008

Thomas M. Coester, Reg. No. 39,637

1279 Oakmead Parkway
Sunnyvale, CA 94085-4040
(310) 207-3800

CERTIFICATE OF ELECTRONIC FILING

I hereby certify that this paper is being transmitted online via EFS Web to the Patent and Trademark Office, Commissioner for Patents, Post Office Box 1450, Alexandria, Virginia 22313-1450, on 4-30, 2008.

Melissa Stead

Melissa Stead 4-30, 2008

VIII. CLAIMS APPENDIX

The claims involved in this Appeal are:

1. (Previously Presented) A system comprising:

a cluster having a first instance and a second instance, each of the first and second instances including a plurality of server nodes;

a control logic to start each instance by initiating a launch logic for each of the server nodes, the launch logic, when initiated, to execute Java processes in each respective server node; and

a communication interface coupled between the launch logic and the control logic to enable the launch logic to obtain status of each of the Java processes and enable the control logic to access the status in a shared memory via the communication interface, the launch logic to store and maintain the status in the shared memory via the communication interface.

2. (Original) The system of claim 1, wherein the launch logic is provided to load a virtual machine and execute a Java process in the virtual machine.

3. (Previously Presented) The system of claim 2, wherein the communication interface comprises:

the shared memory to store the status of the Java processes.

4. (Original) The system of claim 3, wherein the launch logic comprises:

a Java native interface to obtain the status of each of the Java processes and to update the shared memory with the obtained status.

5. (Original) The system of claim 4, wherein the control logic accesses the shared memory to monitor the status of each of the Java processes.

6. (Original) The system of claim 1, wherein the control logic is provided to detect a failure of a Java process and to automatically restart the failed Java process.

7. (Original) The system of claim 1, wherein the control logic is provided to generate an instruction to start, terminate or restart a particular process executed server nodes based on a command received from a remote device.

8. (Original) The system of claim 1, wherein the communication interface further comprises:

a named pipe to send and receive commands between the control logic and the launch logic.

9. (Original) The system of claim 1, wherein the control logic comprises:

a signal handler to receive and interpret signals from a management console.

10. (Original) The system of claim 1, wherein the control logic comprises:

a server connector to enable connection with an external server.

11. (Original) The system of claim 1, wherein the control logic comprises:

Java native processes.

12. (Original) The system of claim 1, wherein the launch logic comprises:

a container combining Java native processes with a Java virtual machine.

13. (Previously Presented) A method comprising:

executing Java processes for a plurality of server nodes in an instance;

obtaining status regarding the Java processes executed by the server nodes in the instance;

storing the status regarding the Java processes in a communication interface, the communication interface updating and maintaining the status in a shared memory;

accessing the status in the shared memory via the communication interface.

14. (Original) The method of claim 13, further comprising:

enabling control of the Java processes based on an instruction received from a remote device.

15. (Original) The method of claim 13, further comprising:

using a Java native interface to obtain the status regarding the Java processes.

16. (Original) The method of claim 13, further comprising:

detecting a failure of a process within the cluster by accessing the status in the communication interface; and

restarting the failed process.

17. (Original) A machine-readable medium that provides instructions, which when executed by a processor cause the processor to perform operations comprising:

executing Java processes for a plurality of server nodes in an instance;

obtaining status regarding each of the Java processes executed by the server nodes in the instance; and

storing the status regarding the Java processes into a shared memory.

18. (Original) The machine-readable medium of claim 17, wherein the operations performed by the processor further comprise:

invoking a Java native interface to obtain the status regarding the Java processes.

19. (Original) The machine-readable medium of claim 17, wherein the operations performed by the processor further comprise:

receiving instructions via a communication interface; and

starting, terminating or restarting a process based on the instructions received via the communication interface.

20. (Original) The machine-readable medium of claim 17, wherein the operations further comprise:

detecting a failure of a process within the cluster by accessing the status in the shared memory and automatically restarting the failed process.

21. (Previously Presented) An apparatus comprising:

a cluster having a first instance and a second instance, each of the first and second instances including a plurality of server nodes;

a control logic to start each respective instance by initiating a launch logic for each respective server node in the first and second instances;

the launch logic, for each respective server node in the first and second instances, to further launch Java processes, and obtain a status of the Java processes to store and maintain in a shared memory; and

the control logic to access the status obtained by the launch logic via the shared memory.

22. (Previously Presented) The apparatus of claim 21, further comprising:

the shared memory to enable exchange of information between the Java processes and the control logic.

23. (Original) The apparatus of claim 21, wherein the launch logic loads a virtual machine and executes Java processes.

24. (Original) The apparatus of claim 23, wherein the launch logic uses a Java native interface to obtain a status of each of the Java processes, and updates information contained in the shared memory based on the status obtained by the Java native interface.

25. (Original) The apparatus of claim 21, wherein the control logic detects a failure of a process within the cluster; and automatically restarts operations of the failed process.

26. (Original) The apparatus of claim 21, further comprising:

a signal handler to receive a command from a remote device and controlling one of the Java processes based on the command received from the remote device.

27. (Original) The apparatus of claim 21, further comprising:

a named pipe to send and receive commands between the control logic and the launch logic.

28. (Previously Presented) A system comprising:

a cluster having a first instance and a second instance, each of the first and second instances including a plurality of server nodes;

means for starting each instance by executing Java processes in each respective server node; and

means for enabling exchange of information that is stored and maintained in a shared memory between the Java processes and the means for starting each instance.

29. (Original) The system of claim 28, further comprising:

means for loading a virtual machine and execute a Java process in the virtual machine.

30. (Previously Presented) The system of claim 28, wherein the means for enabling exchange of information comprises:

the shared memory having a plurality of entries.

31. (Original) The system of claim 30, further comprising:

means for obtaining status for each of the Java processes; and

means for updating the shared memory with the obtained status.

32. (Original) The system of claim 31, further comprising:

means for accessing the shared memory to monitor the status of each of the Java processes; and

means for sending an instruction to the launch means to start, terminate or restart a particular process executed in the cluster.

33. (Original) The system of claim 28, further comprising:

means for enabling a user to monitor and control the Java processes running in the cluster from a management console coupled to the means for controlling; and

means for enabling a connection with an external server.

34. (Previously Presented) The system of claim 1, wherein the launch logic stores the status independent of the control logic accessing the status.

35. (Previously Presented) The system of claim 1, wherein a persistent data structure is stored in the shared memory to enable an independent exchange of information between the Java processes.

36. (Previously Presented) The method of claim 13, wherein accessing the status occurs independent of storing the status.

37. (Previously Presented) The machine-readable medium of claim 17, wherein storing the status occurs independent of accessing the status:

38. (Previously Presented) The apparatus of claim 21, wherein the control logic accesses the status independent of the launch logic storing the status.

39. (Previously Presented) The system of claim 28, wherein the exchange of information is achieved by an independent storing and an independent accessing of the information.

IX. EVIDENCE APPENDIX

"independent." Dictionary.com. 23 April 2008.
<http://dictionary.reference.com/search?q=independent>.

X. RELATED PROCEEDINGS APPENDIX

There are no other appeals or interferences that will directly affect, be directly affected by, or have a bearing on the Board's decision in this appeal.